SLAC
NATIONAL ACCELERATOR LABORATORY

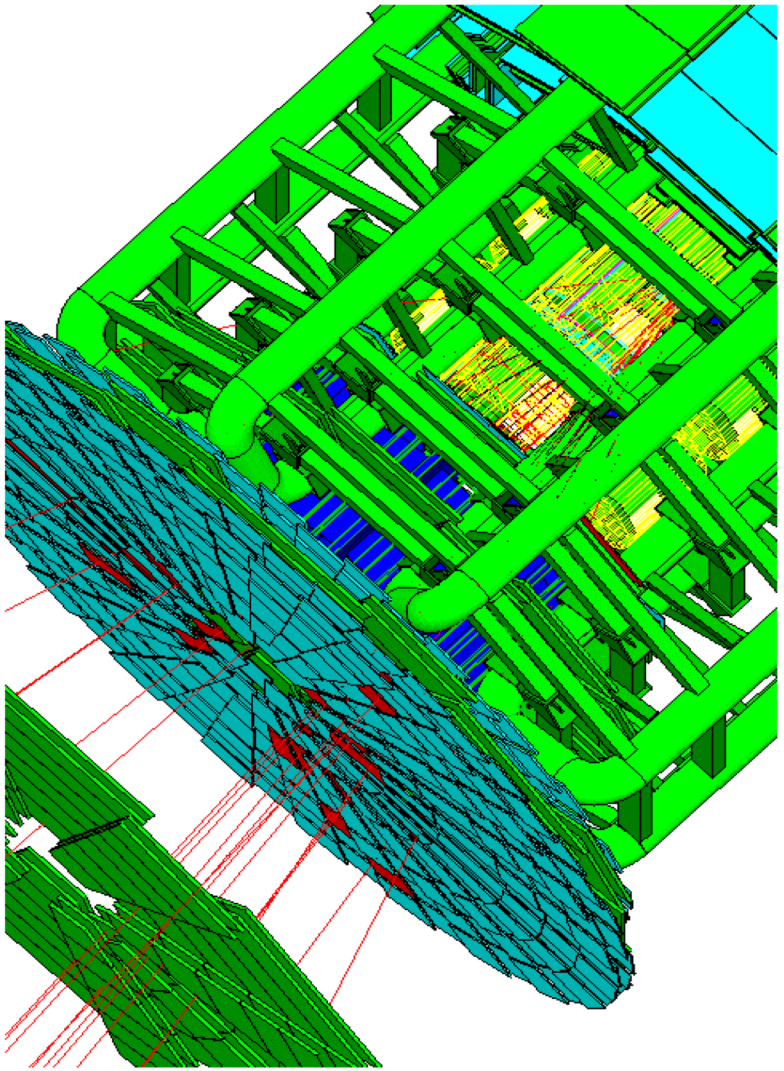
Geometry III

Joseph Perl (SLAC)
Presenting slides by Makoto Asai (SLAC)
Geant4 Tutorial Course

Geant 4

Contents

- Magnetic and other fields
- Nested parameterization
- Assembly volume
- Reflected volume
- Geometry checking tools
- Geometry optimization



SLAC
NATIONAL ACCELERATOR LABORATORY

Defining a magnetic field

Geant 4

Magnetic field (1)

- Create your Magnetic field class
  - Uniform field :
    - Use an object of the G4UniformMagField class

```
G4MagneticField* magField =
    new G4UniformMagField(G4ThreeVector(1.*Tesla,0.,0.);
```

  - Non-uniform field :
    - Create your own concrete class derived from G4MagneticField and implement **`GetFieldValue`** method.

```
void MyField::GetFieldValue(
      const double Point[4], double *field) const
```

    - Point[0..2] are position in global coordinate system, Point[3] is time
    - field[0..2] are returning magnetic field

Other types of field

- The user can create their own type of field, inheriting from G4VField, and an associated Equation of Motion class (inheriting from G4EqRhs) to simulate other types of fields. Field can be time-dependent.
- For pure electric field, Geant4 has G4ElectricField and G4UniformElectricField classes. For combined electromagnetic field, Geant4 has G4ElectroMagneticField class.
- Equation of Motion class for electromagnetic field is G4MagElectricField.

```
G4ElectricField* fEMfield
    = new G4UniformElectricField( G4ThreeVector(0., 100000.*kilovolt/cm, 0.) );
G4EqMagElectricField* fEquation = new G4EqMagElectricField(fEMfield);
G4MagIntegratorStepper* fStepper = new G4ClassicalRK4( fEquation, nvar );
G4FieldManager* fFieldMgr
    = G4TransportationManager::GetTransportationManager()-> GetFieldManager();
fFieldManager->SetDetectorField( fEMfield );
G4MagInt_Driver* fIntgrDriver
    = new G4MagInt_Driver(fMinStep, fStepper,
                                    fStepper->GetNumberOfVariables() );
G4ChordFinder* fChordFinder = new G4ChordFinder(fIntgrDriver);
```

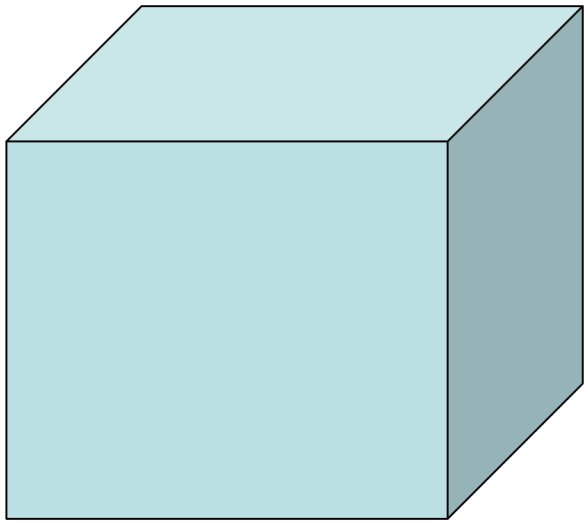
SLAC
NATIONAL ACCELERATOR LABORATORY

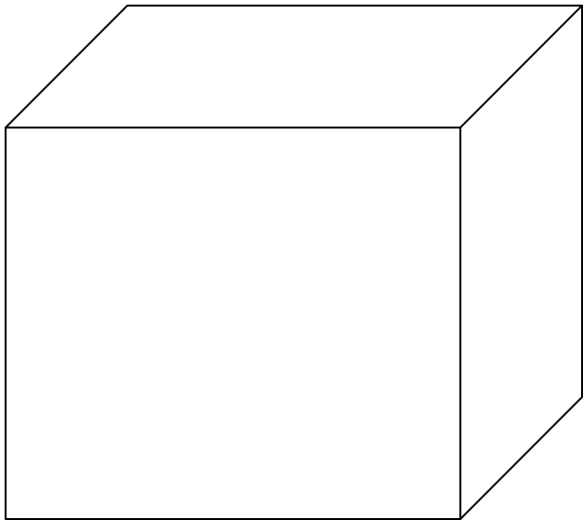
Nested parameterization

Geant 4

Nested parameterization

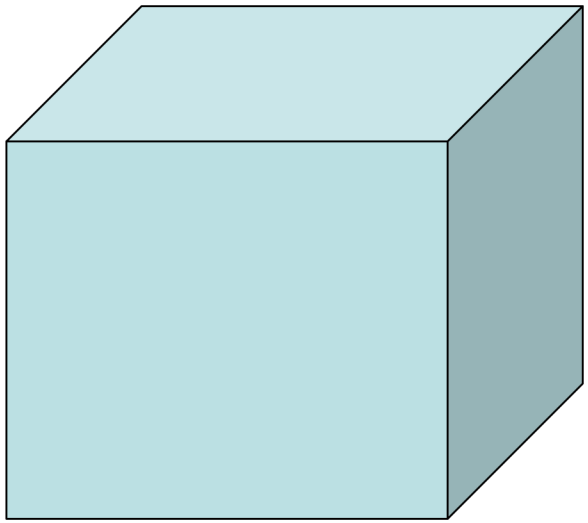
- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)

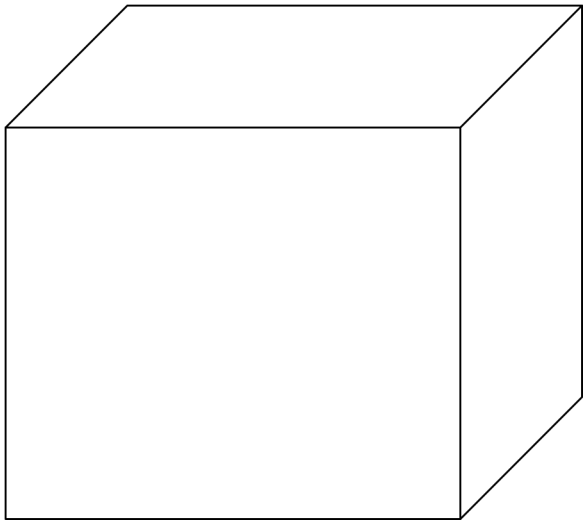




Nested parameterization

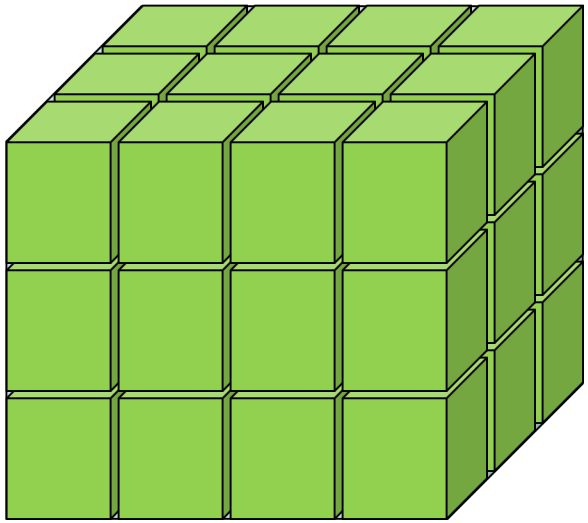
- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

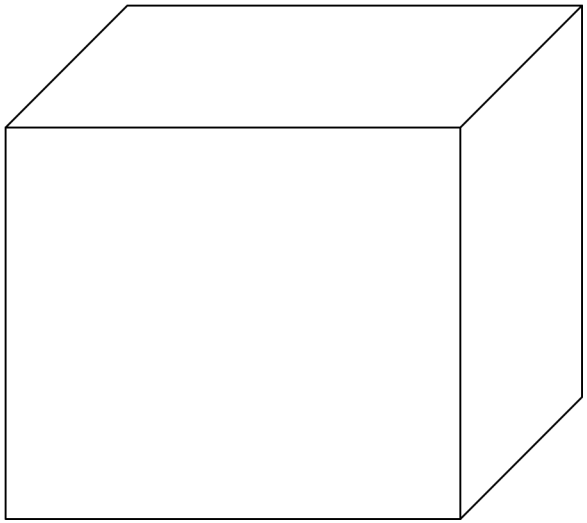




Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

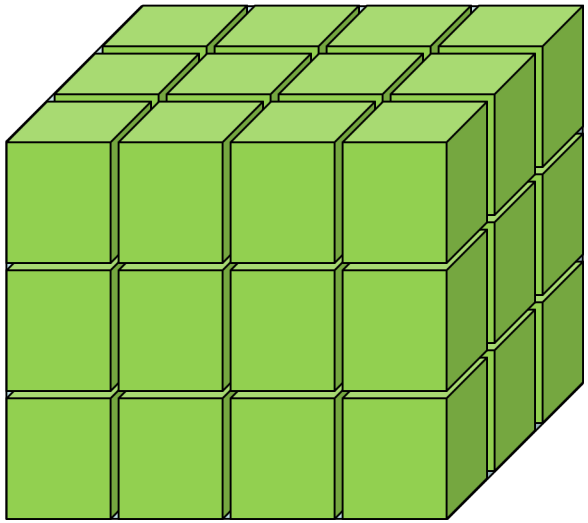


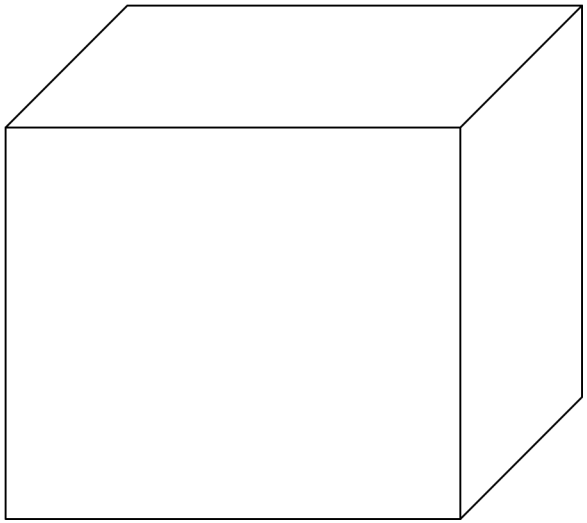


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.

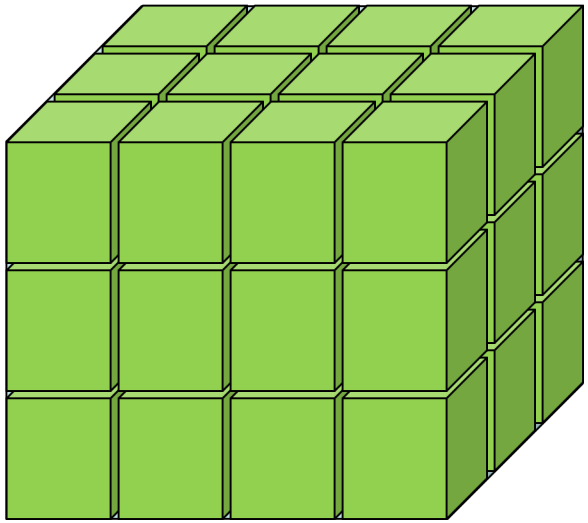


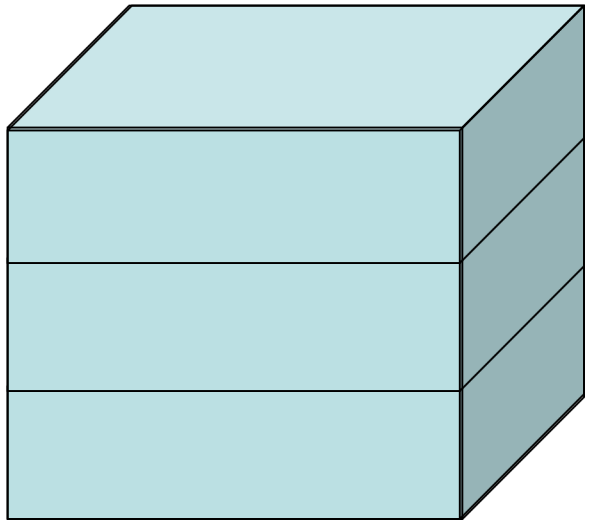


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.

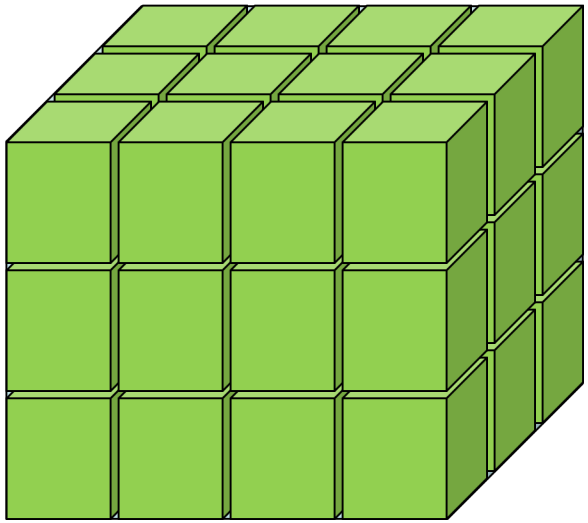


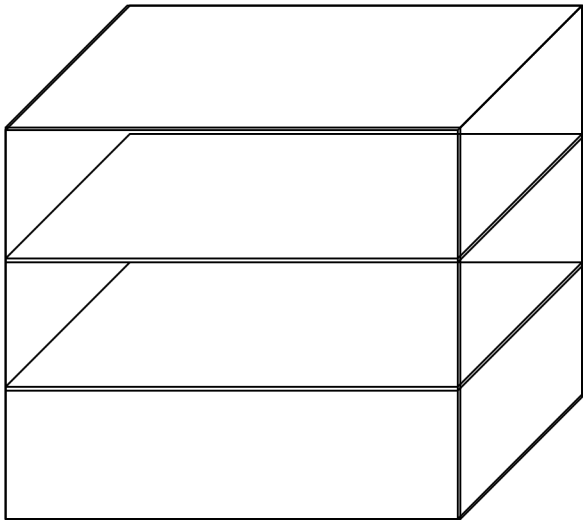


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.

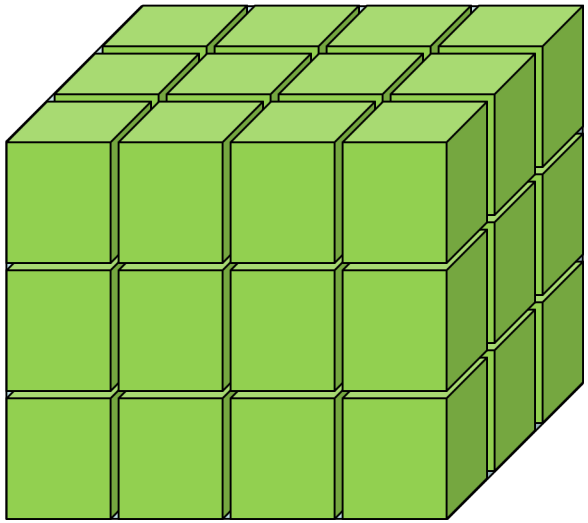


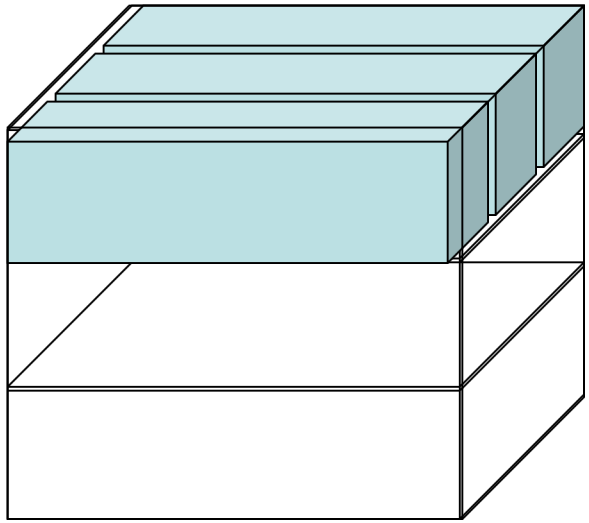


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.

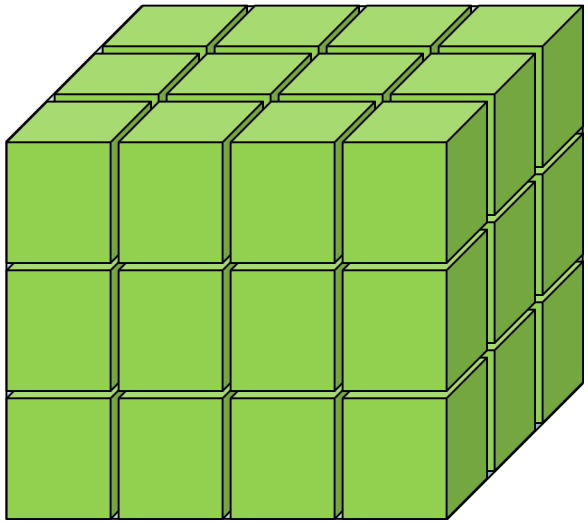


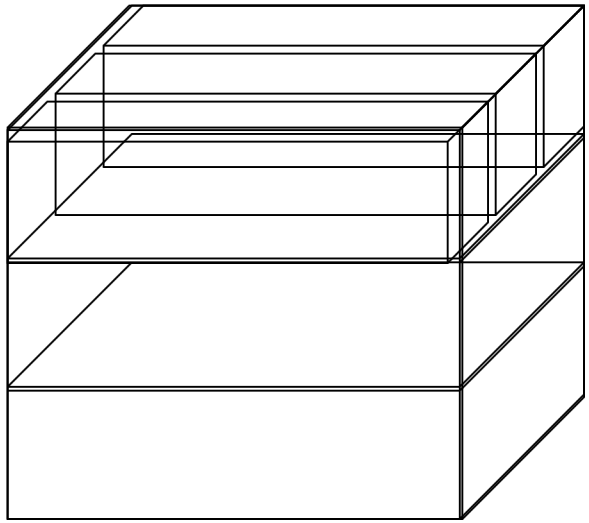


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.

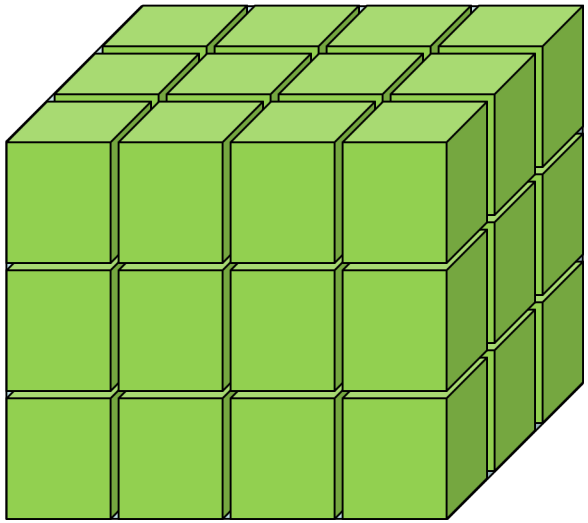


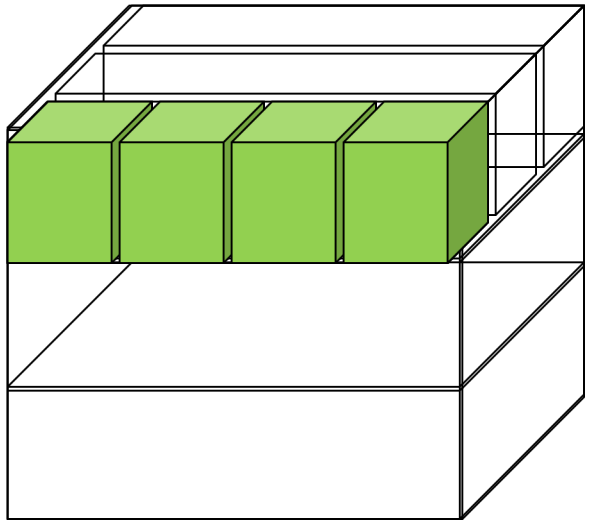


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.

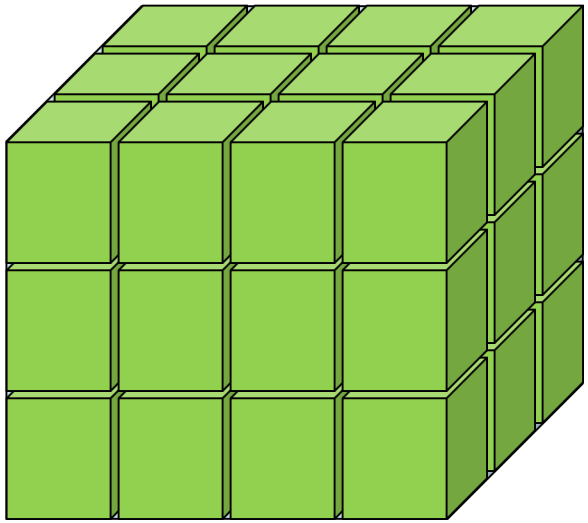


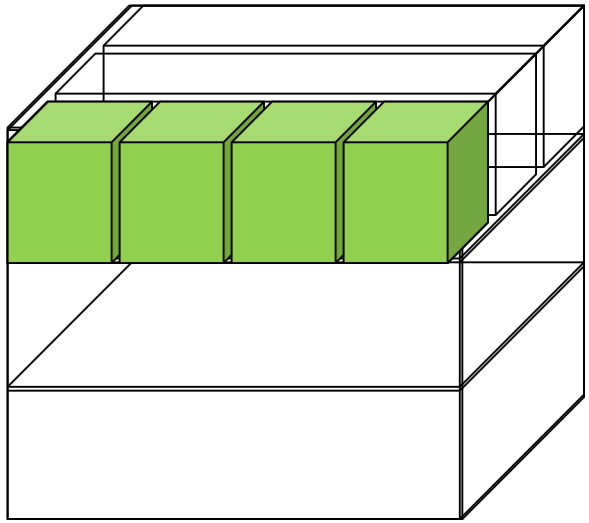


Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

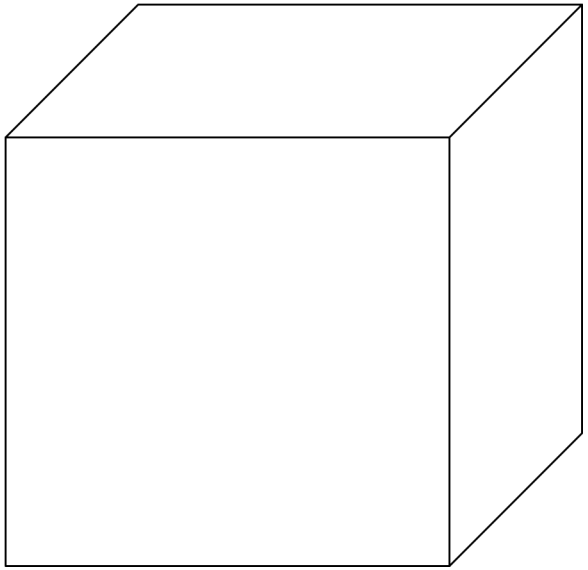
  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.





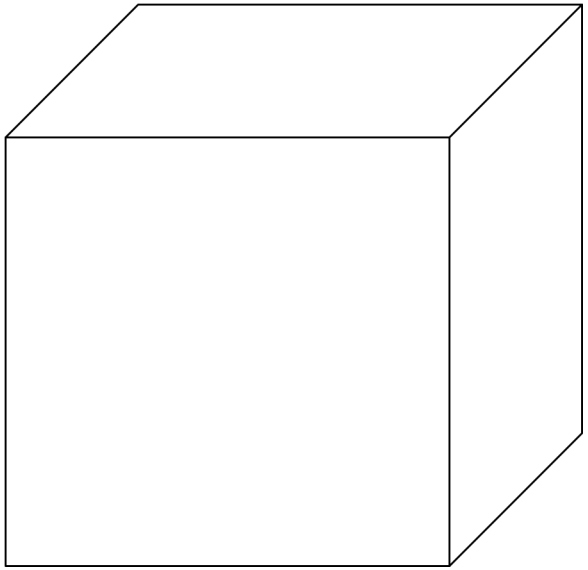
- It requires much less memory for geometry optimization and gives much faster navigation for ultra-large number of voxels.

Nested parameterization



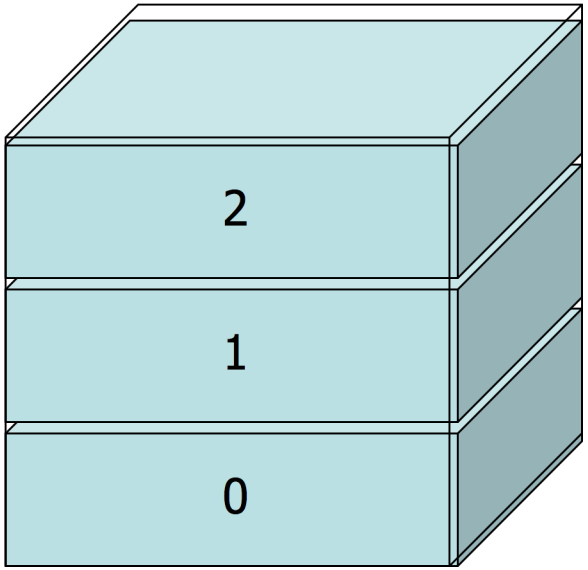
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,



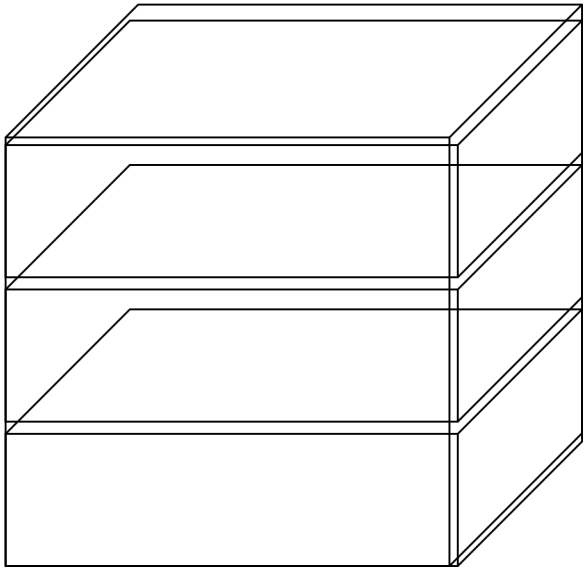
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,



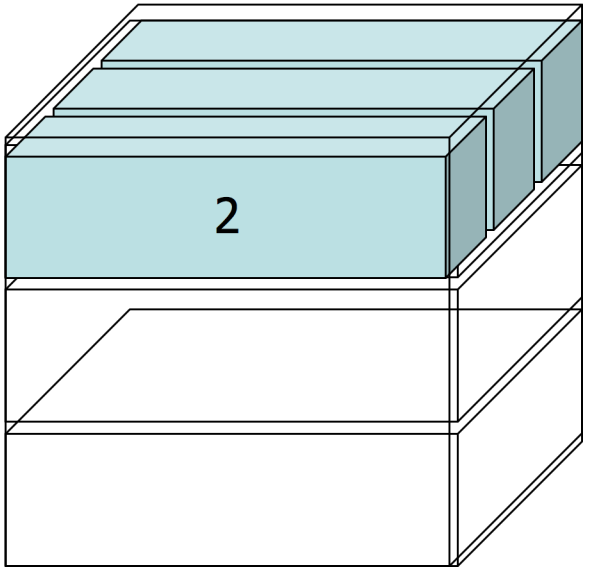
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,



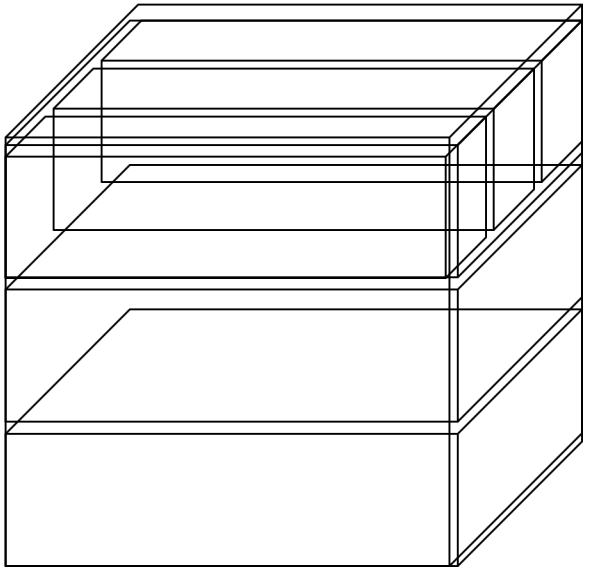
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,



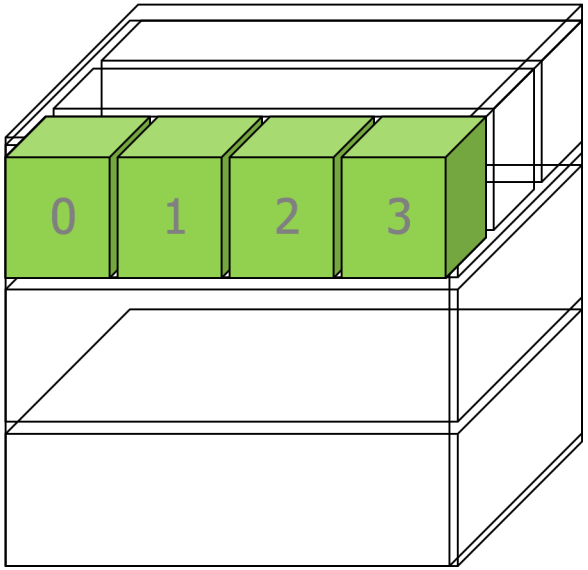
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,



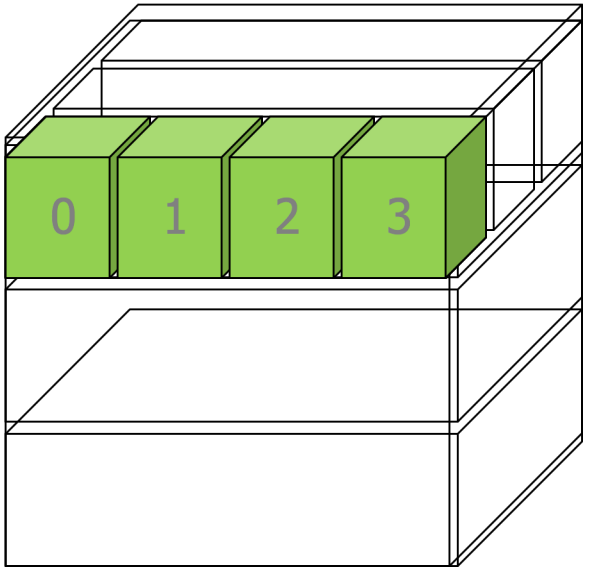
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,



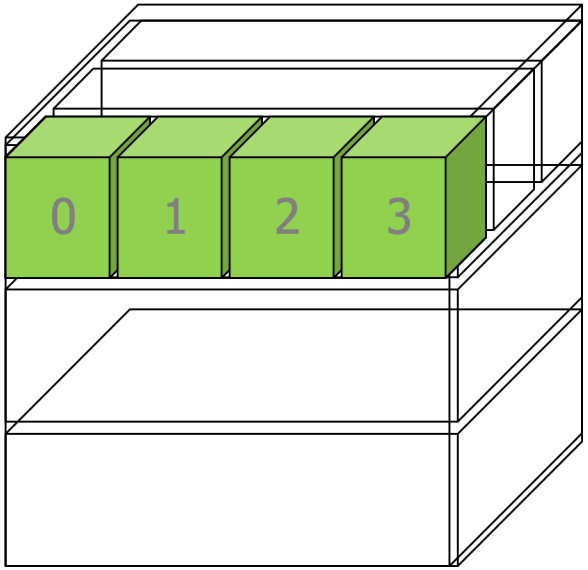
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,
  - Material of a voxel must be parameterized not only by the copy number of the voxel, but also by the copy numbers of ancestors.



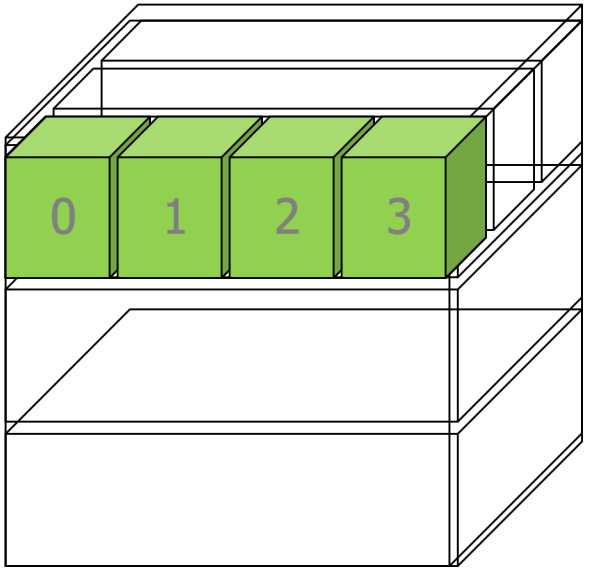
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,
  - Material of a voxel must be parameterized not only by the copy number of the voxel, but also by the copy numbers of ancestors.
  - Material is indexed by three indices.



Nested parameterization

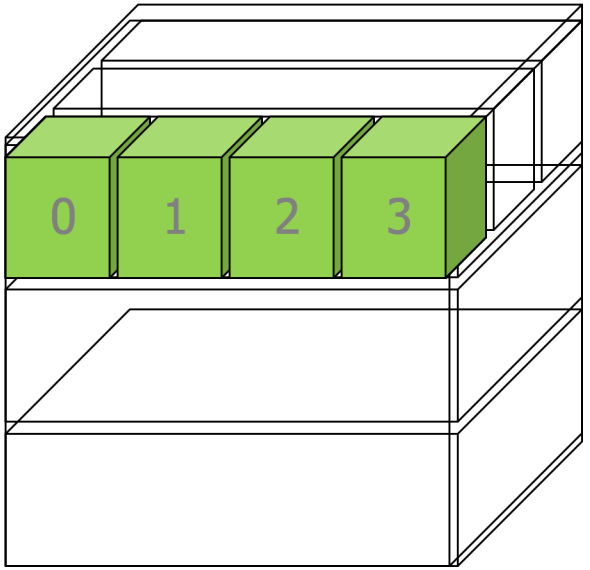
- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,
  - Material of a voxel must be parameterized not only by the copy number of the voxel, but also by the copy numbers of ancestors.
  - Material is indexed by three indices.



- G4VNestedParameterisation is a special parameterization class derived from G4VPVParameterisation base class.

Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,
  - Material of a voxel must be parameterized not only by the copy number of the voxel, but also by the copy numbers of ancestors.
  - Material is indexed by three indices.



- G4VNestedParameterisation is a special parameterization class derived from G4VPVParameterisation base class.
  - ComputeMaterial() method of G4VNestedParameterisation has a touchable object of the parent physical volume, in addition to the copy number of the voxel.
    - Index of first axis = theTouchable->GetCopyNumber(1);
    - Index of second axis = theTouchable->GetCopyNumber(0);
    - Index of third axis = copy number

G4VNestedParameterisation

- G4VNestedParameterisation is derived from G4VPVParameterization.
- G4VNestedParameterisation class has three pure virtual methods you have to implement,
  - in addition to ComputeTransformation() method, which is mandatory for all G4VPVParameterization classes.

virtual G4Material* ComputeMaterial(G4VPhysicalVolume *currentVol,
const G4int repNo, const G4VTouchable *parentTouch=0)=0;

- Return a material pointer w.r.t. copy numbers of itself and ancestors.
- Must cope with parentTouch=0 for navigator's sake. Typically, return a default material if parentTouch=0.

virtual G4int GetNumberOfMaterials() const=0;

- Return total number of materials which may appear as the return value of ComputeMaterial() method.

virtual G4Material* GetMaterial(G4int idx) const=0;

- Return idx-th material.
- “idx” is not a copy number. idx = [0, nMaterial-1]

G4VNestedParameterisation

- G4VNestedParameterisation is a kind of G4VPVParameterization.
  - It can be used as an argument of G4PVParameterised.
  - All other arguments of G4PVParameterised are unaffected.
- Nested parameterization of placement volume is not supported.
  - All levels used as indices of material must be repeated volume. There cannot be a level of placement volume in between.

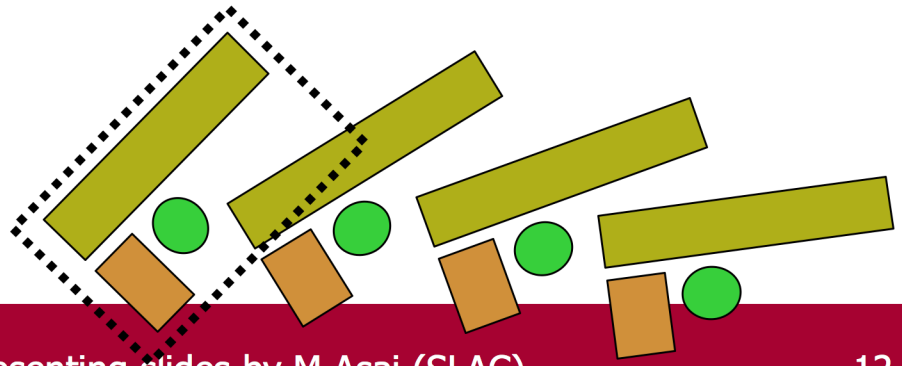
SLAC
NATIONAL ACCELERATOR LABORATORY

Assembly volume

Geant 4

Grouping volumes

- To represent a regular pattern of positioned volumes, composing a more or less complex structure
  - structures which are hard to describe with simple replicas or parameterised volumes
  - structures which may consist of different shapes
  - Too densely positioned to utilize a mother volume
- Assembly volume
  - acts as an *envelope* for its daughter volumes
  - its role is over once its logical volume has been placed
  - daughter physical volumes become independent copies in the final structure
- Participating daughter logical volumes are treated as triplets
  - logical volume
  - translation w.r.t. envelop
  - rotation w.r.t. envelop



G4AssemblyVolume

```
G4AssemblyVolume::AddPlacedVolume
                    ( G4LogicalVolume* volume,
                      G4ThreeVector& translation,
                      G4RotationMatrix* rotation );
```

- Helper class to combine daughter logical volumes in arbitrary way
  - Imprints of the assembly volume are made inside a mother logical volume through `G4AssemblyVolume::MakeImprint(…)`
  - Each physical volume name is generated automatically
    - Format: **av**_WWW_**impr**_XXX_YYY_ZZZ
      - **WWW** – assembly volume instance number
      - **XXX** – assembly volume imprint number
      - **YYY** – name of the placed logical volume in the assembly
      - **ZZZ** – index of the associated logical volume
  - Generated physical volumes (and related transformations) are automatically managed (creation and destruction)

G4AssemblyVolume : example

G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
```

G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
```

G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
```

G4AssemblyVolume : example

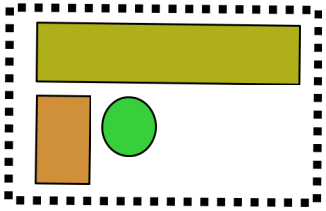
```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
```

G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
```

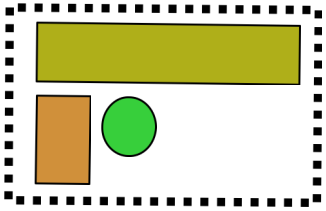
G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
  … // repeat placement for each daughter
```



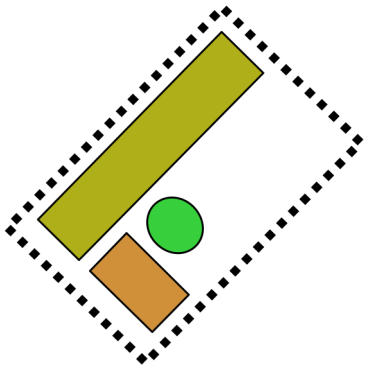
G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
  … // repeat placement for each daughter

for( unsigned int i = 0; i < layers; i++ ) {
  G4RotationMatrix Rm(…);
  G4ThreeVector Tm(…);
  assembly->MakeImprint( worldLV, Tm, Rm );
}
```



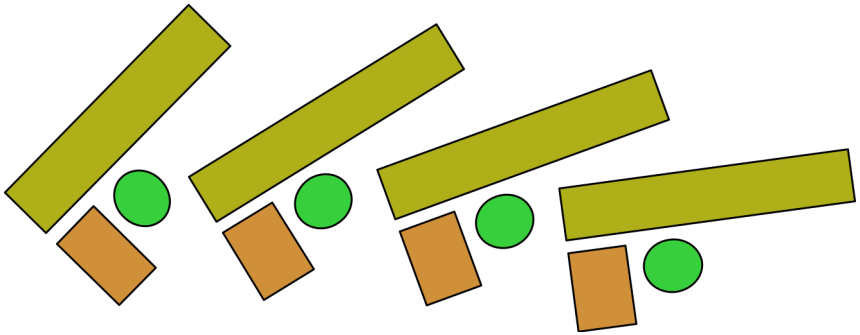
G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
  … // repeat placement for each daughter

for( unsigned int i = 0; i < layers; i++ ) {
  G4RotationMatrix Rm(…);
  G4ThreeVector Tm(…);
  assembly->MakeImprint( worldLV, Tm, Rm );
}
```



G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
  … // repeat placement for each daughter

for( unsigned int i = 0; i < layers; i++ ) {
  G4RotationMatrix Rm(…);
  G4ThreeVector Tm(…);
  assembly->MakeImprint( worldLV, Tm, Rm );
}
```



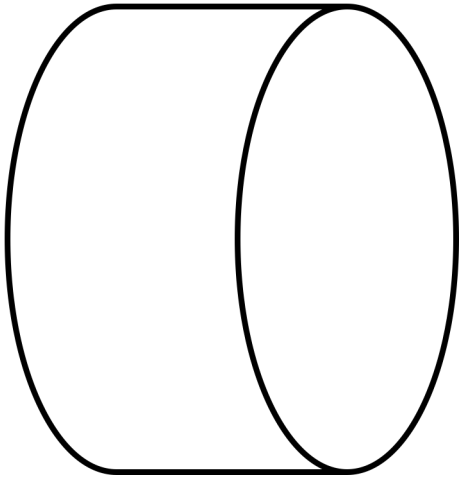
SLAC
NATIONAL ACCELERATOR LABORATORY

Reflected volume

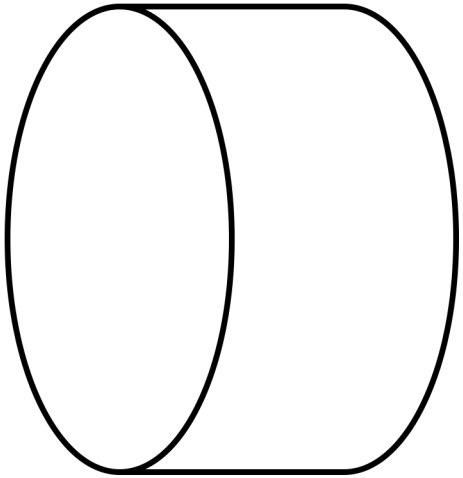
Geant 4

Reflecting solids

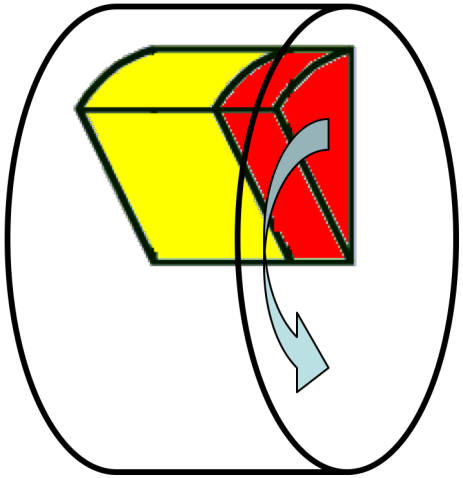
Reflecting solids



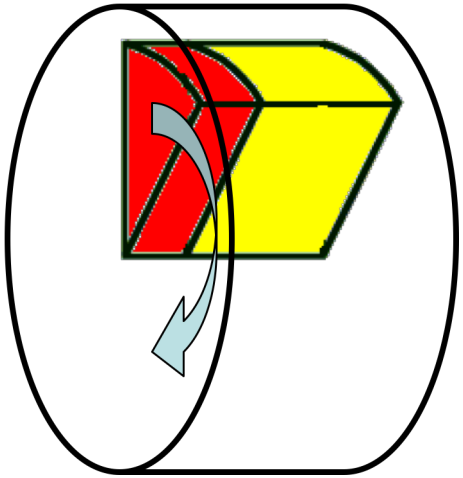
- Let's take an example of a pair of mirror symmetric volumes.



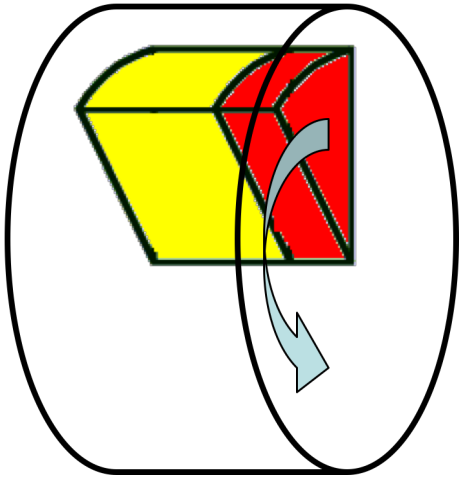
Reflecting solids



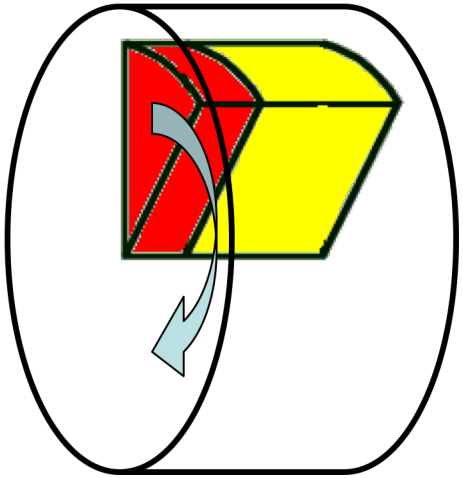
- Let's take an example of a pair of mirror symmetric volumes.



Reflecting solids

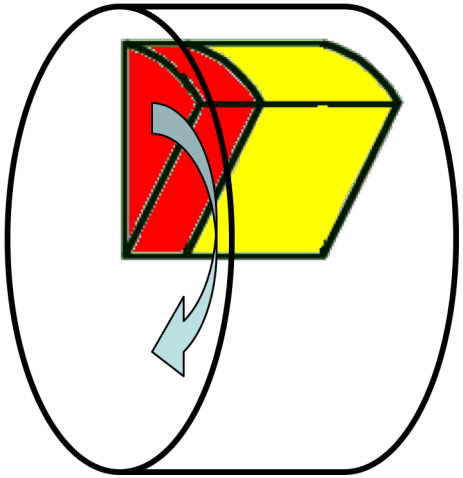


- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation



Reflecting solids

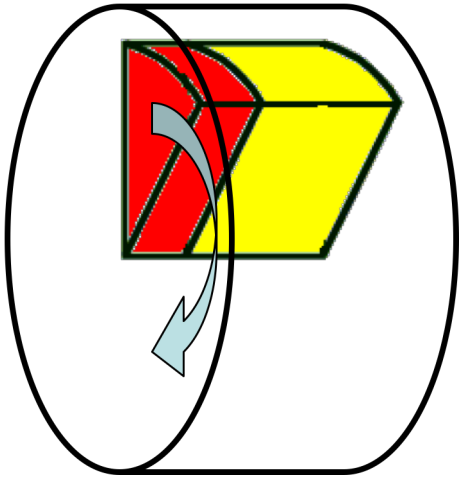
- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation



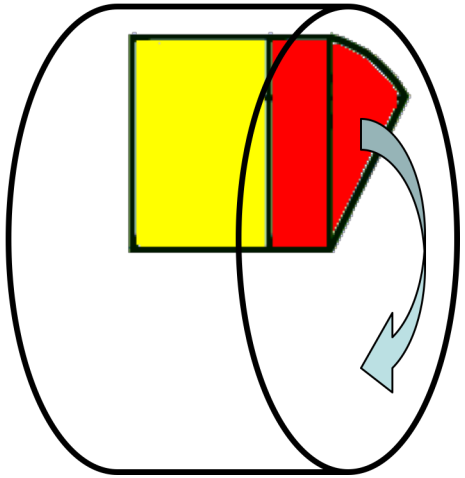
Reflecting solids

- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation

  or 180 degree rotation.



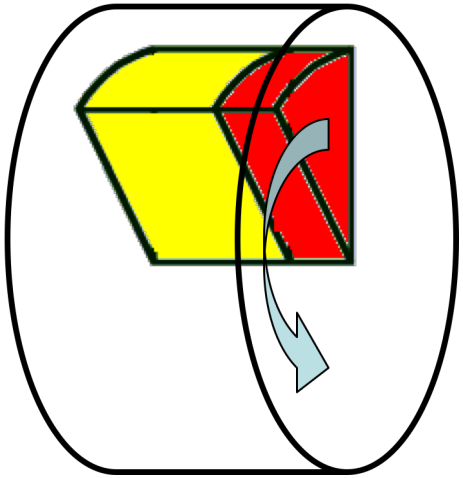
Reflecting solids



- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation

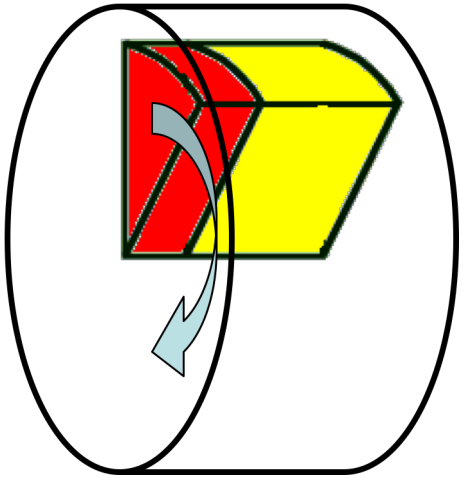
or 180 degree rotation.

Reflecting solids

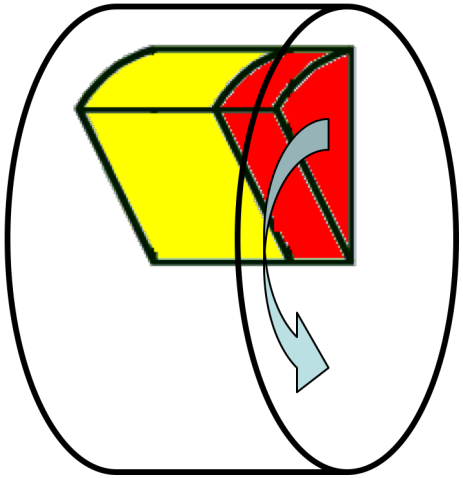


- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation

or 180 degree rotation.

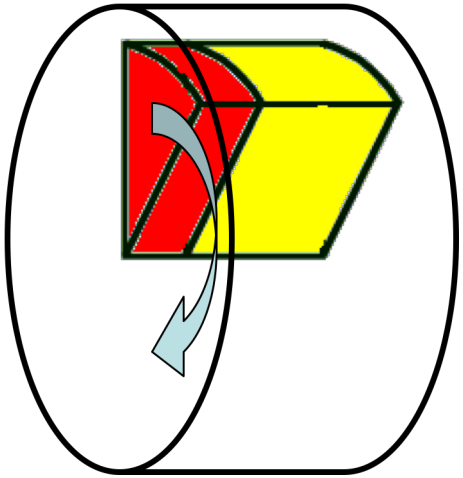


Reflecting solids



- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation

  or 180 degree rotation.



- G4ReflectedSolid (derived from G4VSolid)
  - Utility class representing a solid shifted from its original reference frame to a new mirror symmetric one
  - The reflection (G4Reflect[X/Y/Z]3D) is applied as a decomposition into rotation and translation
- G4ReflectionFactory
  - Singleton object using G4ReflectedSolid for generating placements of reflected volumes
- Reflections are currently limited to simple CSG solids.
  - will be extended soon to all solids

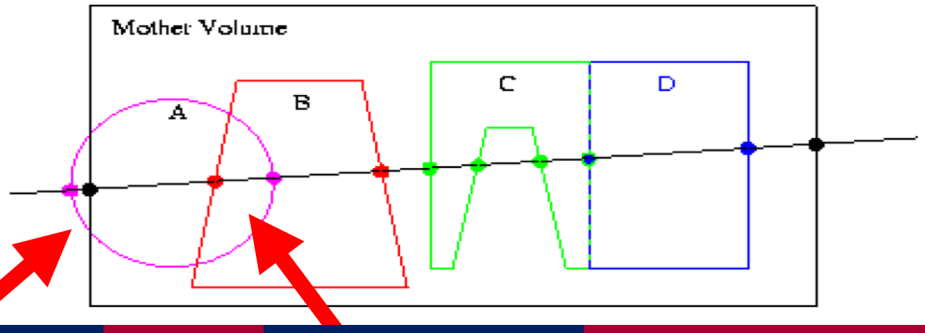
SLAC
NATIONAL ACCELERATOR LABORATORY

Geometry checking tools

Geant 4

Debugging geometries

- An protruding volume is a contained daughter volume which actually protrudes from its mother volume.
- Volumes are also often positioned in a same volume with the intent of not provoking intersections between themselves. When volumes in a common mother actually intersect themselves are defined as overlapping.
- Geant4 does not allow for malformed geometries, neither protruding nor overlapping.
  - The behavior of navigation is unpredictable for such cases.
- The problem of detecting overlaps between volumes is bounded by the complexity of the solid models description.
- Utilities are provided for detecting wrong positioning
  - Optional checks at construction
  - Kernel run-time commands
  - Graphical tools (DAVID, OLAP)



Optional checks at construction

- Constructors of G4PVPlacement and G4PVParameterised have an optional argument “pSurfChk”.

```
G4PVPlacement(G4RotationMatrix* pRot,
   const G4ThreeVector &tlate,
   G4LogicalVolume *pDaughterLogical,
   const G4String &pName,
   G4LogicalVolume *pMotherLogical,
   G4bool pMany, G4int pCopyNo,
   G4bool pSurfChk=false);
```

- If this flag is true, overlap check is done at the construction.
  - Some number of points are randomly sampled on the surface of creating volume.
  - Each of these points are examined
    - If it is outside of the mother volume, or
    - If it is inside of already existing other volumes in the same mother volume.
- This check requires lots of CPU time, but it is worth to try at least once when you implement your geometry of some complexity.

Debugging run-time commands

- Built-in run-time commands to activate verification tests for the user geometry are defined
  - to start verification of geometry for overlapping regions based on a standard grid setup, limited to the first depth level

    `geometry/test/run` or **`geometry/test/grid_test`**
  - applies the grid test to all depth levels (may require lots of CPU time!)

    `geometry/test/recursive_test`
  - shoots lines according to a cylindrical pattern

    `geometry/test/cylinder_test`
  - to shoot a line along a specified direction and position

    `geometry/test/line_test`
  - to specify position for the **`line_test`**

    `geometry/test/position`
  - to specify direction for the **`line_test`**

    `geometry/test/direction`

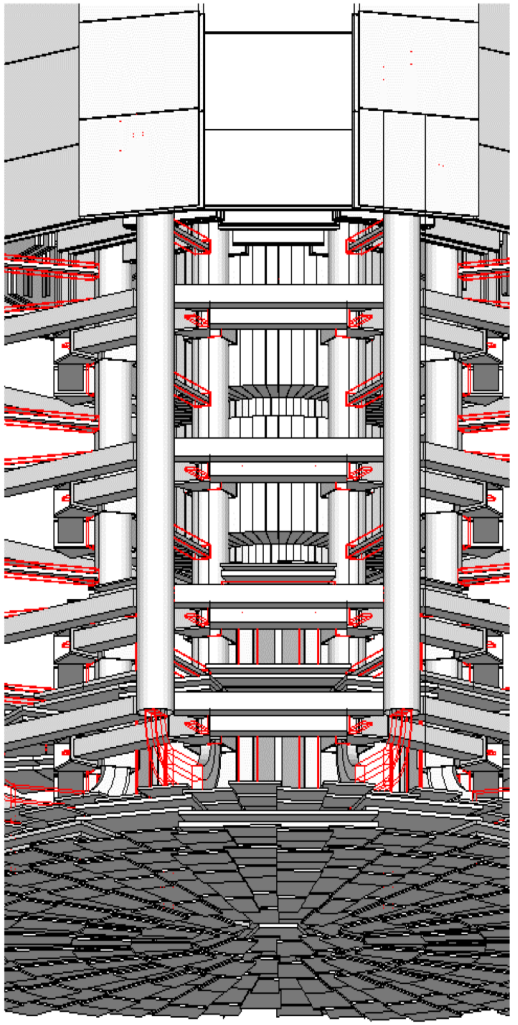
Debugging run-time commands

- Example layout:

```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list
   truncated)
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    240           -240      -145.5      -145.5     0       -145.5      -145.5
Which in the mother coordinate system are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
Which in the coordinate system of Tracker[0] are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
Which in the coordinate system of Overlap[0] are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
```

Debugging tools: DAVID

- DAVID is a graphical debugging tool for detecting potential intersections of volumes
- Accuracy of the graphical representation can be tuned to the exact geometrical description.
  - physical-volume surfaces are automatically decomposed into 3D polygons
  - intersections of the generated polygons are parsed.
  - If a polygon intersects with another one, the physical volumes associated to these polygons are highlighted in color (red is the default).
- DAVID can be downloaded from the Web as external tool for Geant4
  - http://geant4.kek.jp/~tanaka/



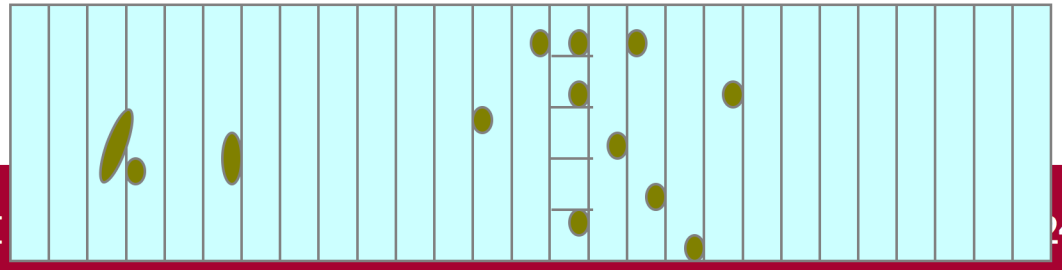
SLAC
NATIONAL ACCELERATOR LABORATORY

Geometry optimization ("voxelization")

Geant 4

Smart voxelization

- In case of Geant 3.21, the user had to carefully implement his/her geometry to maximize the performance of geometrical navigation.
- While in Geant4, user's geometry is automatically optimized to most suitable to the navigation. - "Voxelization"
  - For each mother volume, one-dimensional virtual division is performed.
  - Subdivisions (slices) containing same volumes are gathered into one.
  - Additional division again using second and/or third Cartesian axes, if needed.
- *"Smart voxels"* are computed at initialisation time
  - When the detector geometry is *closed*
  - Does not require large memory or computing resources
  - At tracking time, searching is done in a hierarchy of virtual divisions



Detector description tuning

- Some geometry topologies may require ‘special’ tuning for ideal and efficient optimisation
  - for example: a dense nucleus of volumes included in very large mother volume
- Granularity of voxelisation can be explicitly set
  - Methods **`Set/GetSmartless()`** from **`G4LogicalVolume`**
- Critical regions for optimisation can be detected
  - Helper class **`G4SmartVoxelStat`** for monitoring time spent in detector geometry optimisation
    - Automatically activated if **`/run/verbose`** greater than `1`

```
Percent     Memory     Heads     Nodes    Pointers     Total CPU    Volume
-------     ------     -----     -----    --------     ---------    -----------
  91.70         1k         1        50          50          0.00    Calorimeter
   8.30         0k         1         3           4          0.00    Layer
```

Visualising voxel structure

- The computed voxel structure can be visualized with the final detector geometry
  - Helper class **`G4DrawVoxels`**
  - Visualize voxels given a logical volume

    ```
    G4DrawVoxels::DrawVoxels(const G4LogicalVolume*)
    ```

  - Allows setting of visualization attributes for voxels

    ```
    G4DrawVoxels::SetVoxelsVisAttributes(…)
    ```

  - useful for debugging purposes